constructing a control flow graph / representation (CFG) of the programming code based on the identified basic blocks and identified links, the CFG being in a preliminary form;

identifying at least one basic block ending in a dynamic branch;

exploring, based on the CFG, and without information from a source external to the first compiled code state of the programming code, all identified basic blocks that lead to the dynamic branch as far back as is necessary to fully determine a set of destination addresses for the dynamic branch, the set of destination addresses defining the set of destinations from the dynamic branch;

examining the set of destinations to identify a branch table;

updating the CFG to reflect the set of destinations and the identified branch table; and

translating the programming code from the first compiled code state to the second compiled code state based at least in part on the updated CFG.

REMARKS

Claims 1-39 were initially pending in the parent application. Claims 4-13, 17-26 and 30-39 have been canceled herein, having been allowed in the parent application. Claims 1-3, 14-16 and 27-29 were finally rejected by the Examiner in the parent application and remain in the current application.

Applicants have amended independent claims 1, 14 and 27 herein. Applicants respectfully traverse the Examiner's rejection set forth in the Final Office Action dated June 29, 2001. Applicants further respectfully submit that the present application is now in condition for allowance, as the prior art does not teach or suggest the features recited in Applicants' claims for the reasons set forth below. Reconsideration of the Office Action and an early Notice of Allowance are respectfully solicited.

The Examiner states that the International Search Report listed in the "Other Document" section of the Information Disclosure Statement filed on February 22, 2001 will not be considered. Applicants respectfully note that the International search report was provided merely as a courtesy to the Examiner and that all the references cited in the International Search Report

were separately listed in the Information Disclosure Statement. Therefore the Examiner is again requested to consider the International Search Report.

Rejection of Claims 1-3, 14-16, and 27-29 under 35 U.S.C. §102(b)

The Examiner has rejected claims 1-3, 14-16, and 27-29 under 35 U.S.C. §102(b) as being anticipated by Muchnick and has rejected claims 1-2, 14-15 and 27-28 under 35 U.S.C. §102(b) as being anticipated by Banning et al., PCT WO 9001738. Applicants respectfully traverse the §102(b) rejections for the following reasons.

Claim 1 recites a method of translating compiled programming code from a first compiled code state to a second compiled code state. The programming code in the first compiled code state includes a number of basic blocks. Each basic block includes a set of instructions. At least one of the blocks ends in a dynamic branch. A dynamic branch is a transfer to one of a set of destinations. The destination is based on a calculation of a destination address. To calculate the destination address, basic blocks included in the first compiled code state of the programming code are identified. Links between the basic blocks are identified. A preliminary control flow graph (CFG) of the programming code based on the identified basic blocks and links is constructed. At least one block ending in a dynamic branch is identified. Without information from an external source, the basic blocks leading to the dynamic branch are traced back to determine a set of destination addresses for the dynamic branch. A branch table is built from the set of destinations and the CFG is updated to reflect the set of destinations and the identified branch table. The programming code is translated from the first compiled code state to the second compiled code state based at least in part on the updated CFG.

I. The Muchnick Reference

Muchnick discloses compiling a non-compiled high level programming language into compiled code. In contrast, Applicants' invention claims re-compiling already-compiled (executable) code into a second form of compiled (executable) code. Amended independent claim 1 recites a method of translating programming code from a first *compiled* code state to a second *compiled* code state and in particular, a method step of translating programming code from the first compiled code state to the second compiled code state. The input to Muchnick's compiler is source code (not executable code) while the input to Applicants' invention is compiled

(executable) code. Source code, unlike compiled (executable) code, is not an end product; that is, a computer cannot execute source code. In contrast, compiled code is an end product, and can be executed by a computer. Thus the Muchnick reference doesn't anticipate claim 1.

Furthermore, the problem addressed by Applicants' invention would not arise when compiling source code and thus is not addressed or handled in Muchnick. When translating from a first compiled code state to a second compiled code state via a re-compiler or translator, a problem arises where the machine code in the first compiled state includes branch or destination address information, where such branch address information would be different from the machine code in the second state, and where *explicit address information from the original compiler is absent*. Obviously, this problem does not arise in a compiler that compiles source code into executable code, because "explicit address information from the original compiler" is not absent. There is only one compiler.

Independent claims 14 and 27 recite essentially the same subject matter, although in the form of a translator and a computer readable medium, respectively.

Thus, the Muchnick reference does not anticipate claims 1, 14, or 27 or any depending claims including claims 2-3, 15-16, and 28-29 and Applicants respectfully request reconsideration and withdrawal of the §102(b) rejection with respect to the Muchnick reference.

II.   The Banning Reference

Banning discloses a machine process in which a first program in a first binary language is translated into a second program in a second binary language. "The first steps of the algorithm... are to read the input data necessary to carry out the represented process. This data includes the source program 108 in 8086 binary machine language and any application specific data (asd) 112 associated with the source binary program." (See Banning, page 6 lines 11-16).

In contrast, Applicants' invention specifically addresses the problem that arises when such application specific data is not available. In particular, claim 1 recites:

> "exploring, based on the CFG, and *without information from a source external to the first compiled code state of the programming code,* all identified basic blocks that lead to the dynamic branch as far back as

is necessary to fully determine a set of destination addresses for the
dynamic branch, the set of destination addresses defining the set of
destinations from the dynamic branch;".

Because such external information is available in Banning in the application specific data
that is input in addition to the source program (see Banning, page 6, lines 11-16), the problem
addressed by Applicants' invention is not addressed or handled in Banning.

When translating from a first compiled code state to a second compiled code state via a
re-compiler or translator, the machine code in the first compiled state includes branch or
destination address information. This branch address information would be different from the
machine code in the second state, and where *explicit address information from the original
compiler is absent,* the correct address must be otherwise obtained.

Applicants addressed the problem of acquiring correct branch or destination address
information by exploring all the basic blocks leading to the dynamic branch based on a control
flow graph (CFG), *as far back as is necessary to fully determine a set of destination addresses*
for the dynamic branch, as recited in claim 1. Back tracking analysis is performed by creating a
tree containing alias nodes and links and traversing the tree in reverse chronological order. A
branch table is built and the CFG is updated to reflect the set of destinations and the identified
branch table. The programming code is translated from the first code state to the second code
state, based at least in part on the updated CFG. Applicants' claim 1 has been amended to
state:

"exploring, based on the CFG, and without information from a
source external to the first compiled code state of the programming
code, all identified basic blocks that lead to the dynamic branch as
far back as is necessary to fully determine a set of destination
addresses for the dynamic branch, the set of destination addresses
defining the set of destinations from the dynamic branch;"

in order to clarify the above.

Independent claims 14 and 27 recite essentially the same subject matter, although in the
form of a translator and a computer readable medium, respectively and have been similarly
amended.

Thus, the Banning reference does not anticipate claims 1, 14, or 27 or any of the dependent claims including claims 2-3, 15-16, and 28-29; therefore, Applicants respectfully request reconsideration and withdrawal of the §102(b) rejection with respect to the Banning reference.
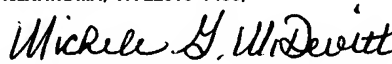
In view of the foregoing amendments and remarks, Applicants respectfully submit that the present application, including claims 1-3, 14-16 and 27-29, is in condition for allowance, as the prior art does not teach or suggest the features recited in Applicants' claims for the reasons set forth above. Reconsideration of the Office Action and an early Notice of Allowance are respectfully solicited. Should the Examiner be of the opinion that a telephone or other interview would expedite the processing of this application, he/she is invited to contact the undersigned at (215) 986-5169. Should the Examiner have any other questions he or she is also invited to contact the undersigned at the same number.

Respectfully submitted,

ise A. Rode
Reg. No. 37,226

Unisys Corporation
Unisys Way
Blue Bell, Pennsylvania 19424-0001

-8-

<u>VERSION WITH MARKINGS TO SHOW CHANGES MADE</u>

TITLE OF THE INVENTION
Determining Destinations of a Dynamic Branch

INVENTORS
G. Lawrence Krablin

Andrew T. Jennings

Timothy N. Fender

William Stratton

CROSS-REFERENCE TO RELATED APPLICATIONS

<u>This application is a continuation of U.S. Serial No. 09/239,282 filed January 29, 1999 entitled "Determining Destinations of a Dynamic Brach," the entire disclosure of which is incorporated by reference.</u>

FIELD OF THE INVENTION

The present invention relates to determining destinations of a dynamic branch in compiled computer code (i.e., machine code). More particularly, the present invention relates to determining every possible destination of such dynamic branch so that the control flow of such computer code with respect to such dynamic branch may be accurately determined.

BACKGROUND OF THE INVENTION

In certain circumstances, it is useful to translate compiled machine code from a first state corresponding to a first instruction set to a second state corresponding to a second instruction set. As it is known, such translating is preferable to the alternative: locating the source code from which the machine code in the first code state was derived;

TITLE OF THE INVENTION
Determining Destinations of a Dynamic Branch

INVENTORS
G. Lawrence Krablin

Andrew T. Jennings

Timothy N. Fender

William Stratton

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a continuation of U.S. Serial No. 09/239,282 filed January 29, 1999 entitled "Determining Destinations of a Dynamic Brach," the entire disclosure of which is incorporated by reference.

FIELD OF THE INVENTION

The present invention relates to determining destinations of a dynamic branch in compiled computer code (i.e., machine code). More particularly, the present invention relates to determining every possible destination of such dynamic branch so that the control flow of such computer code with respect to such dynamic branch may be accurately determined.

BACKGROUND OF THE INVENTION

In certain circumstances, it is useful to translate compiled machine code from a first state corresponding to a first instruction set to a second state corresponding to a second instruction set. As it is known, such translating is preferable to the alternative: locating the source code from which the machine code in the first code state was derived;

1.	(Amended) A method of translating compiled programming code from a first <u>compiled</u> code state to a second <u>compiled</u> code state, the

5	programming code in the first <u>compiled</u> code state comprising a plurality of basic blocks, each basic block comprising a set of instructions, at least one basic block ending in a dynamic branch, the dynamic branch being a transfer to one of a set of destinations based on a calculation of a destination address, the method comprising the steps of:

10	identifying the plurality of basic blocks in the first <u>compiled</u> code state of the programming code;

identifying links between the identified basic blocks;

constructing a control flow graph / representation (CFG) of the programming code based on the identified basic blocks and identified links,

15	the CFG being in a preliminary form;

identifying at least one basic block ending in a dynamic branch;

exploring, based on the CFG, <u>without information from a source external to the first compiled code state of the programming code,</u> all

20	identified basic blocks that lead to the dynamic branch as far back as is necessary to fully determine a set of destination addresses for the dynamic branch, the set of destination addresses defining the set of destinations from the dynamic branch;

examining the set of destinations to identify a branch table;

25	updating the CFG to reflect the set of destinations and the identified branch table; <u>and</u>

translating the programming code from the first <u>compiled</u> code state to the second <u>compiled</u> code state based at least in part on the updated CFG.

30

14.	(Amended) A translator operating on a processor for translating compiled programming code from a first <u>compiled</u> code state to a second <u>compiled</u> code state, the programming code in the first <u>compiled</u> code state comprising a plurality of basic blocks, each basic block comprising a set of

instructions, at least one basic block ending in a dynamic branch, the dynamic branch being a transfer to one of a set of destinations based on a calculation of a destination address, the translator:

        identifying the plurality of basic blocks in the first <u>compiled</u> code

5   state of the programming code;

        identifying links between the identified basic blocks;

        constructing a control flow graph / representation (CFG) of the programming code based on the identified basic blocks and identified links, the CFG being in a preliminary form;

10   identifying at least one basic block ending in a dynamic branch;

        exploring, based on the CFG, <u>and without information from a source external to the first compiled code state of the programming code,</u> all identified basic blocks that lead to the dynamic branch as far back as is

15   necessary to fully determine a set of destination addresses for the dynamic branch, the set of destination addresses defining the set of destinations from the dynamic branch;

        examining the set of destinations to identify a branch table;

        updating the CFG to reflect the set of destinations and the

20   identified branch table; <u>and</u>

        translating the programming code from the first <u>compiled</u> code state to the second <u>compiled</u> code state based at least in part on the updated CFG.

25   27.   (Amended) In connection with translating compiled programming code from a first <u>compiled</u> code state to a second <u>compiled</u> code state, the programming code in the first <u>compiled</u> code state comprising a plurality of basic blocks, each basic block comprising a set of instructions, at least one basic block ending in a dynamic branch, the

30   dynamic branch being a transfer to one of a set of destinations based on a calculation of a destination address, a computer- readable medium having computer-executable instructions for performing steps comprising:

        identifying the plurality of basic blocks in the first <u>compiled </u>code state of the programming code;

identifying links between the identified basic blocks;

constructing a control flow graph / representation (CFG) of the programming code based on the identified basic blocks and identified links, the CFG being in a preliminary form;

5   identifying at least one basic block ending in a dynamic branch;

exploring, based on the CFG, <u>and without information from a source external to the first compiled code state of the programming code,</u> all identified basic blocks that lead to the dynamic branch as far back as is

10 necessary to fully determine a set of destination addresses for the dynamic branch, the set of destination addresses defining the set of destinations from the dynamic branch;

examining the set of destinations to identify a branch table;

updating the CFG to reflect the set of destinations and the

15 identified branch table; <u>and</u>

translating the programming code from the first <u>compiled</u> code state to the second <u>compiled</u> code state based at least in part on the updated CFG